MINI PROJECT 2: TYPE CHECKER - TOPIC 3

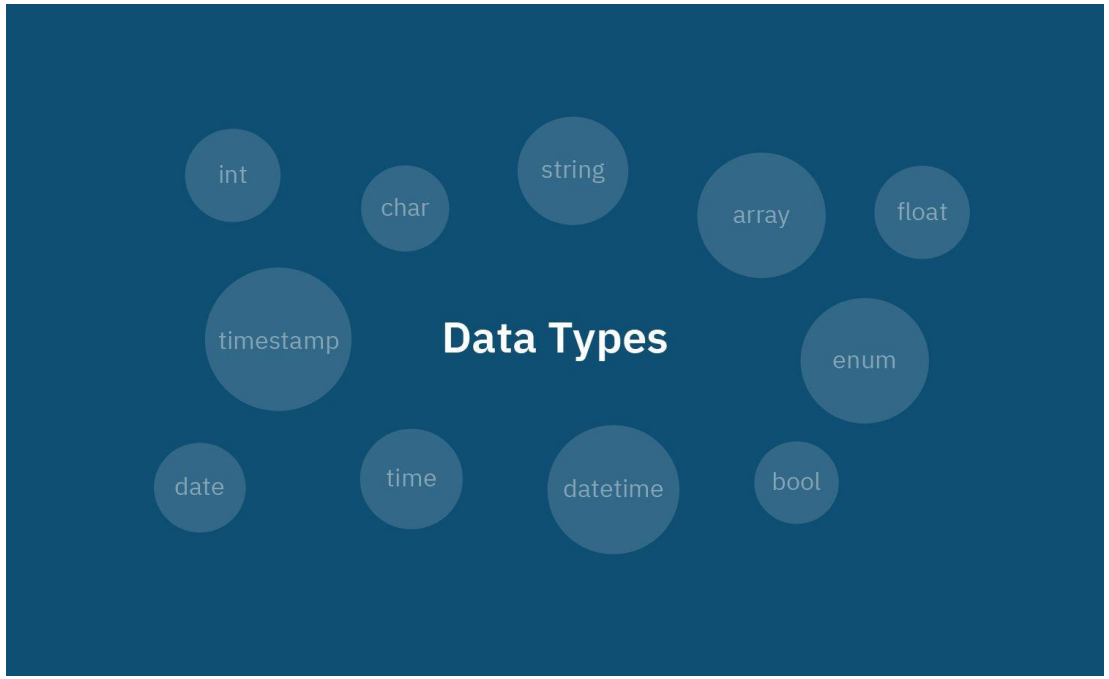
MATTHEW GOTTLIEB, DANIEL DEWAR, VLADIMIR NICOLAS, AUSTIN NORMAN, BASSIL GHAREEB

THE PROBLEM

- Generally, errors only show up when a program runs and it crashes.
- That also means that a bug might be waiting to be found as you had to access a rarely used branch just to get it to run
- How would you address something like that?
- What if we could catch an error before running into anything?

WHAT IS A TYPE

- Simple. A type is a label, a classification of what a certain piece of data represents.
- Num is a number
- (Num -> Num) is a function



- For our purposes, we don't actually care about the value
- All were doing is keeping the category

WHAT DOES A TYPE CHECKER DO?

- Before
  - parse -> desugar -> interp
- After
  - Parse -> desugar -> **type-check** -> interp
    - The type-checker exists to ensure a possible path in all branches.
    - The type checker is almost like a second interpreter, but instead of computing values it computes types.

WHAT DID WE DO

- We added a few things
- The Type datatype – labels for numbers and functions
- Type annotation on lambdas - {lam x body} to {lam {x : Num} body}
  - This is just telling the type checker what type the parameter is supposed to be
- The type environment – exactly like the value environment but stores "x is Num" instead of "x is 5"
- The type-check function – Same recursive structure as the interpreter, walks the same AST, but returns types instead of values. Every expression form has one rule – a number is always **`numT`**, addition requires both sides to be **`numT`**, and so on.
- Slot the type-check between desugaring and interp

LIVE EXAMPLE

- Here we go…